Interfacing HTCondor-CE with OpenStack

B. Bockelman[1], J. Caballero Bejar[2], J. Hover [2]

[1] University of Nebraska-Lincoln, Lincoln, NE 68588, USA
[2] Brookhaven National Laboratory, PO BOX 5000 Upton, NY 11973, USA

E-mail: bbockelm@cse.unl.edu, jcaballero@bnl.gov, jhover@bnl.gov

Abstract. Over the past few years, Grid Computing technologies have reached a high level of maturity. One key aspect of this success has been the development and adoption of newer Compute Elements to interface the external Grid users with local batch systems. These new Compute Elements allow for better handling of jobs requirements and a more precise management of diverse local resources.

However, despite this level of maturity, the Grid Computing world is lacking diversity in local execution platforms. As Grid Computing technologies have historically been driven by the needs of the High Energy Physics community, most resource providers run the platform (operating system version and architecture) that best suits the needs of their particular users.

In parallel, the development of virtualization and cloud technologies has accelerated recently, making available a variety of solutions, both commercial and academic, proprietary and open source. Virtualization facilitates performing computational tasks on platforms not available at most computing sites.

This work attempts to join the technologies, allowing users to interact with computing sites through one of the standard Computing Elements, HTCondor-CE, but running their jobs within VMs on a local cloud platform, OpenStack, when needed.

The system will re-route, in a transparent way, end user jobs into dynamically-launched VM worker nodes when they have requirements that cannot be satisfied by the static local batch system nodes. Also, once the automated mechanisms are in place, it becomes straightforward to allow an end user to invoke a custom Virtual Machine at the site. This will allow cloud resources to be used without requiring the user to establish a separate account. Both scenarios are described in this work.

1. Introduction

In parallel to the evolution of the Grid Computing technologies, a pletora of virtualization tools and techniques have been developed, both in the industry and academia worlds. The Grid Computing has reached a high level of maturity on certain aspects, but it lacks some flexibility in others. In particular, the type of flexibility that those virtualization tools can offer. For historical reasons the Grid Computing technology has evolved around the needs of the High Energy Physics (HEP) community, which has a very limited set of contrains, but those are in fact almost unavoidable. One example of these contrains is the need for a specific Linux platform to be deployed on the resources part of the HEP-oriented Grid infrastructre.

This scenario is highly efficient for the HEP experiments, but may present a barrier for other scientific communities.

On the another hand, many scientific and academia facilities offer to their employees -or even to external scientist-, the ability to use virtualization platforms. Several of these platforms have become very mature products over the past years **ADD REFERENCES HERE TO OPENSTACK AND OPENNEBULA, FOR EXAMPLE**
The usage of these virtualized resources eliminates the contrains in the classic Grid infrastructure, but forces the users to learn how to use them, including getting the client tools and ID accounts.

The goal of this work is to bring both worlds, the Grid Computing and the virtualization, closer to each other. We study how to allow users to submit their grid jobs to a standard Compute Element -the HTCondor-CE in this work-, expressing platform requirements that do not necesarily fit the classical HEP-oriented environment. Also, we present a mechanism to let users to interact with a virtualization cluster without requiring special knowledge, install clients, or request authentication credentials.

1.1. Prototype setup

The setup for this prototype was simple. All grid identities where mapped at the CE as a single UNIX account, which also corresponded to the unique OpenStack tenant. The backend batch queue was also HTCondor. The VM images used in OpenStack had an HTCondor startd preconfigured to join the mentioned backup HTCondor pool. These startd daemons were also preconfigured to shut down after the first job finished, in order to prevent them from picking more than one job.
Also, job submission was done one by one.
The technical specifications of the OpenStack cluster used for this prototype are as listed

- OpenStack instance: Icehouse.
- 120 compute nodes (16 cores, 32 GM RAM, 2 to 5 TB disks).
- 200 TB Swift (Amazon S3-equivalent) object store.
- EC2 API enabled.

The version of HTCondor was 8.4.8.

2. Prototype description

In order to allow the HTCondor-CE to dedice when jobs need to be executed on a VM server in OpenStack instead of any of nodes in the backend batch system, we make use of one feature included in the CE: the JobRouter hooks **MAYBE ADD REFERENCE HERE**
The Job Router is, by definition, ad add-on that transform jobs from one type to another according to a configurable policy. The Job Router hooks are arbitrary code that can be invoked at some points during the job life cycle.
?? HOW DO I ADD A CAPTION TO A MINIPAGE OTHER THAN FIGURE ???

```
JOB_ROUTER_HOOK_KEYWORD = NOVA

NOVA_HOOK_TRANSLATE_JOB = /usr/share/virtualgridsite/nova_hook_translate_job
NOVA_HOOK_UPDATE_JOB_INFO = /usr/libexec/nova_hook_update_job_info
NOVA_HOOK_JOB_EXIT = /usr/libexec/nova_hook_job_exit
NOVA_HOOK_JOB_CLEANUP = /usr/libexec/nova_hook_cleanup_job
```

2.1. Case 1: elastic expansion

The first scenario allows running user's job on an OpenStack server when the job's requirements do not match the characteristics of the nodes in the backend batch system. A typical case is when user's job requires and older version of the operative system, or a newer one. Job's requirements are listed in Table 1

job classad	**description**
+opsys	Type of the OS. Example: LINUX
+opsysname	Name of OS. Example: CentOS
+opsysmajorversion	Version of the OS. Example: 7
+maxMemory	Amount of RAM memory needed.
+disk	Hard disk size
+xcount	Number of cores

Table 1: Job's classad

The process is shown in 1. The HTCondor-CE's TRANSLATE hook compares the jobs requirements with a set of configuration files compiling the entire list of host profiles available, both in the backend batch system and VM images ready to be used in OpenStack. As a result of that comparison, when it decides a new VM server is needed to run the job, it requests directly booting it in OpenStack. As mentioned, the VMs are prepared to initialize an HTCondor's startd daemon and join the backend pool. Once the VM booting process has finished, the TRANSLATE hook finalizes and the source job gets routed to the backend pool, with the guarantee that there is now a host that can run it. It also added a new ad-hoc classad to the job with the name assigned to the VM server.

The job execution finalization triggers a call to the CLEANUP hook. This one will noticed the mentioned custom classad, so it can proceed with the termination of the VM server.

2.2. Case 2: interactive usage

In this case, the user knows there is an OpenStack cluster behind the CE, and she actually wants to boot a VM server to login and work interactively. There is no payload to be executed in this case.

This request for an interactive VM is expressed by adding a special job classad:

+virtualgridsite_interactive_vm = true

As can be seen in Figure **??**fig:interactive), when the TRANSLATE hook detects that job classad, it transforms the job in place -so it is not routed into any backend batch queue-, and it is converted into an EC2 job **?? MAYBE ADD HERE A REFERENCE TO THE CONDOR DOC ON EC2 ??**
This works because, as mentioned, the OpenStack infrastructure has the EC2 API enabled. **?? REFERENCE TO THE SECTION WHERE THAT WAS MENTIONED ??**

In this scenario, it is the routed job, now converted into an EC2 job, the one in charge of requesting the VM instantiation in OpenStack. Once that step is completed, the user can read the public IP of that new VM server, and log into it. This IP is available because it is written as a job classad that is set to be mirrored back with the following set

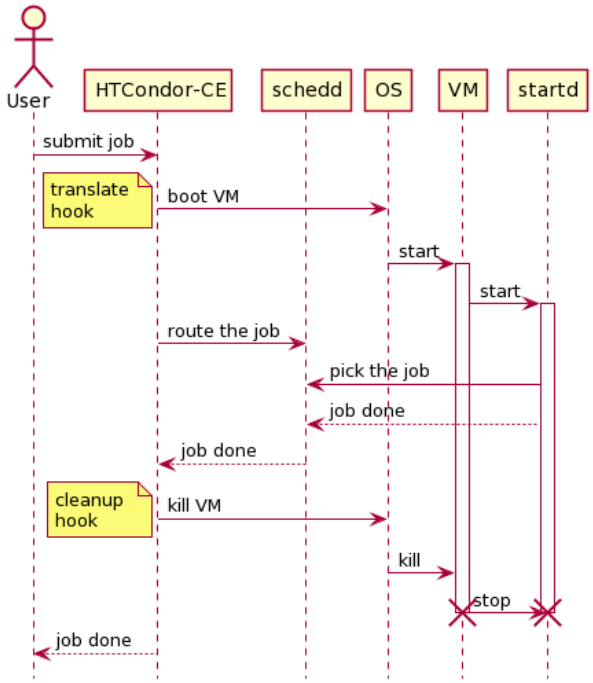


Figure 1: Sequence diagram for the elastic case

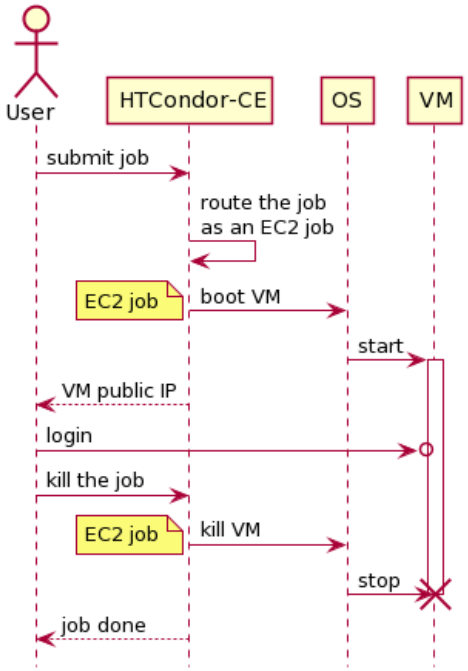


Figure 2: Sequence diagram for the interactive usage case

NOVA_ATTRS_TO_COPY = EC2ElasticIP

Once the user does not have a need for the VM, can issue a *condor_rm* command, which will

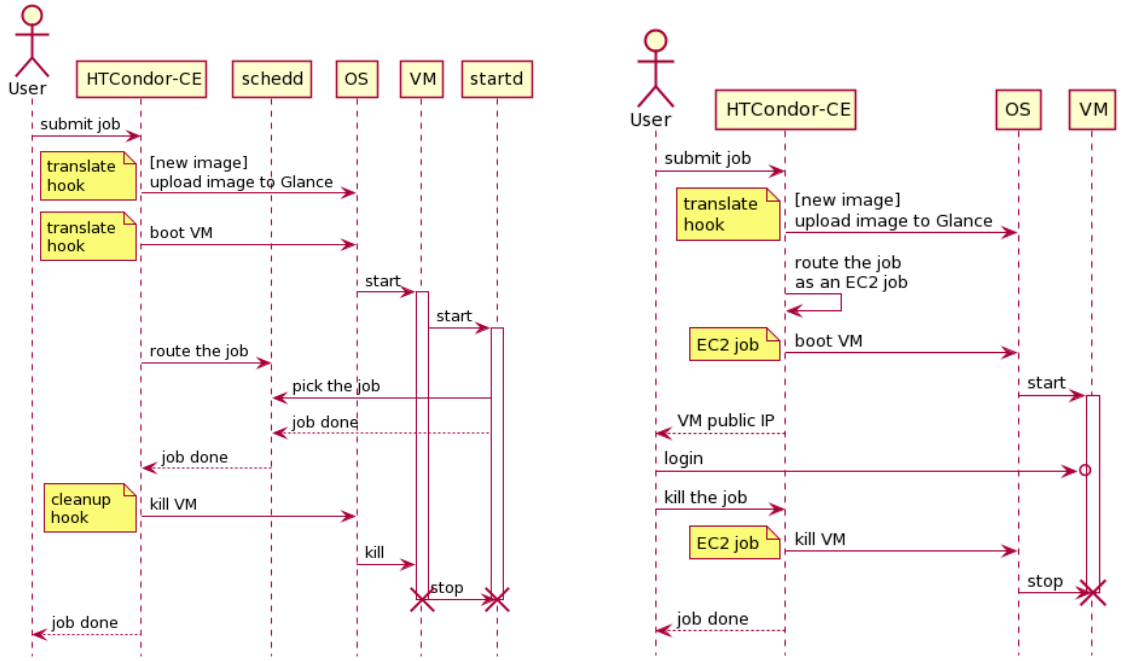
make the job to be completed. Once again, that event will be captured by the CLEANUP hook, which will terminate the VM server.

2.3. Custom virtual machine image

Users can provide the VM image they need themselves. This is done by passing the URL with the image as a job classad:

$$+\text{virtualgridsite\_url} = <\text{URL with the image}>$$

As can be seen in Figure 3, the only difference with respect the previous cases is that the new VM image is uploaded into Glance **REFERENCE HERE TO GLANCE** when needed. This image is uploaded with an unique name, created as a combination of the URL hash and its timestamp. This way it becomes trivial to determine whether the requested image has already being uploaded or not.



(a) elastic case (b) interactive usage

Figure 3: Sequence diagrams when using user's custom image

3. Security

As the configuration files, as well the OpenStack credentials files, are placed on the same CE host, it is important to prevent the user's jobs from running on that host. This is prevented by adding the following setup.

```
START_LOCAL_UNIVERSE = False
START_SCHEDULER_UNIVERSE = $(START_LOCAL_UNIVERSE)
```

4. Problems found

Several limitations were found during this investigation. Some of these limitations are currently being fixed in the HTCondor source code. Deeper understanding of the others may lead to improvements in the overall system.

If the communication process with OpenStack by the TRANSLATE hook fails, there is currently a clean way to terminate the job, or to put it in a HOLD state. Instead, the Job Router tries again, after 30 seconds, in an virtually infinite loop.

For the mirroring of the classads from the routed job back to the source job, it is required the UPDATE hook to send the whole classad to the stdout. It has been found that actions fail due hidden bugs in the code.

it is not easy to manage a job when its requirements not only can not be satisfied by any node in the backend batch system, but also by any currently available VM image. The solution implemented, as the time being, is to detect that case early in the job cycle management, and create a dedicated route for it. To detect this case, we leverage a feature that allows for the partial creation of the routing table by code. This code reads the configuration files with all avaible images types, and builds the logic to identify jobs that do not meet any of those criteria. When this ocurrs, the job is transform in place -and therefore not routed to the backend batch queue-, with an extra classad (set_noroute=True) to prevent it from being considered again for routing. This let the job in permanent IDLE status, candidate for removal if also a periodic_remove expression is added to its classad definition.

```
JOB_ROUTER_ENTRIES_CMD =
      /usr/lib/python2.6/site-packages/virtualgridsite/routes.py
JOB_ROUTER_ENTRIES_REFRESH = 600
```

```
[ Name = "No_route";
EditJobInPlace = true;
set_noreroute = "True";
Requirements = TARGET.noreroute is undefined && <node requirements>;
set_PeriodicRemove = ( JobStatus == 1 && ( time() - EnteredCurrentStatus ) > 10 );
TargetUniverse = 5
]
```

Another problem found is related to the present configuration after installation rather than code design. The HTCondor-CE only allows, by default, jobs being routed in certain ways -more exactly, to certain Job Universes-. It is easy to workaround this constrain by overriden the right configuration variable (JOB_ROUTER_SOURCE_SOURCE_JOB_CONSTRAINT) removing that specific constrain.

```
JOB_ROUTER_SOURCE_JOB_CONSTRAINT =
(target.x509userproxysubject =!= UNDEFINED) &&
(target.x509UserProxyExpiration =!= UNDEFINED) &&
(time() < target.x509UserProxyExpiration)
```

5. Future work

This work can be improved in several ways.

First, it should be straigthforward to expand the same concept to other platforms and not only OpenStack. For example, to AWS **REFERENCE HERE TO EC2 ???**, or remote clusters using the BOSCO-CE mechanism **REFERENCE TO BOSCO-CE**.

The presetup in the VM images should be avoided, allowing the HTCondor startd to join any arbitrary pool.

The job lifecycle management allows for improvements. One possibility is to allow the same VM server to run more than one job, increasing the efficiency of the whole system. Another option is to reuse the presented mechanism to elastically expand the size of the local backend batch queue when the jobs waiting time passes some thredshold.

Having more than just one OpenStack user (tenant) is highly recommendable to create policies on user quotas, fair shares mechanisms, accounting and billing.

Removing the need for GSI authentication to submit jobs to the HTCondor-CE is under consideration.

It is also being studied a change in the architectural design to prevent the hooks code from interacting themselves with OpenStack. Instead, an independent daemon would listen to the hooks requests and execute the corresponding steps. This will prevent from the need to read the configuration files for each job, and also allows for having a single process which holds the state of the entire system, including all jobs, which will facilitate implemented more complex and smarter policies.

Acknowledgments

The authors would like to thank the HTCondor developers for their constant support.

References

[1] Thain D, Tannenbaum T and Livny M 2005 Distributed Computing in Practice: The Condor Experience *Concurrency and Computation: Practice and Experience* **17, No. 2-4** 323-56

[2] Bockelman B, Cartwright T, Frey J, Fajardo E M, Lin B, Selmeci M, Tannenbaum T and Zvada M 2015 Commissioning the HTCondor-CE for the Open Science Grid *Journal of Physics: Conference Series* **664, No. 6** 062003